Package: MadanTextNetwork (via r-universe)

October 4, 2024

Type Package

Title Persian Text Mining Tool for Co-Occurrence Network

Version 0.1.0

Description Provides an extension to 'MadanText' for creating and analyzing co-occurrence networks in Persian text data. This package mainly makes use of the 'PersianStemmer' (Safshekan, R., et al. (2019). <https://CRAN.R-project.org/package=PersianStemmer>), 'udpipe' (Wijffels, J., et al. (2023). <https://CRAN.R-project.org/package=udpipe>), and 'shiny' (Chang, W., et al. (2023). <https://CRAN.R-project.org/package=shiny>) packages.

License GPL-3

Encoding UTF-8

RoxygenNote 7.1.2

Depends R (>= 4.0.0)

Imports xlsx, glue, lattice, stopwords,textmineR, tidytext, tidyr, udpipe, PersianStemmer, shiny (>= 1.8.0), shinythemes, tm, dplyr, hwordcloud, stringr, stringi, topicmodels, igraph, ngram, visNetwork

NeedsCompilation no

Author Kido Ishikawa [aut, cre], Hasan Khosravi [aut]

Maintainer Kido Ishikawa <kido.ishikawa6@gmail.com>

Repository CRAN

Date/Publication 2023-12-08 11:30:05 UTC

Contents

`ASDATA.FRAME` *Convert to Data Frame*

Description

This function converts the given object to a data frame.

Usage

```
ASDATA.FRAME(x)
```

Arguments

`x`	An object to be converted into a data frame.

Value

Returns a data frame with rows and columns corresponding to the original object's structure. If 'x' is a matrix, each column in the matrix becomes a column in the data frame. If 'x' is a list where all elements are of the same length, each element of the list becomes a column in the data frame. Attributes such as rownames, colnames, and dimnames (if any) are preserved in the conversion.

Examples

```
data <- ASDATA.FRAME(matrix(1:4, ncol = 2))
```

cluster.graph *Cluster a Graph and Extract Largest Component*

Description

This function applies clustering to a graph and extracts the largest connected component.

Usage

```
cluster.graph(network)
```

Arguments

network	A graph object.

Value

A list containing three elements: 'gr' with the largest connected component of the graph, 'cl' with a data frame of nodes and their cluster membership, and 'node.impo' with a data frame of node importance measures like degree, closeness, and betweenness.

Examples

```
## Not run:
  # Assuming 'network' is a predefined graph object
  cluster.graph(network)

## End(Not run)
```

Community.Detection.Membership *Get Community Membership of a Graph*

Description

This function applies community detection to a graph and returns the membership information of each node.

Usage

```
Community.Detection.Membership(network)
```

Arguments

network	A graph object.

Value

A data frame where each row represents a node in the graph, with columns for the node name and its corresponding community membership number. This information is useful for understanding the community structure within the graph.

Examples

```
## Not run:
  network <- make_graph("Zachary")
  membership_info <- Community.Detection.Membership(network)
  print(membership_info)

## End(Not run)
```

Community.Detection.Plot

Plot Community Detection in a Graph

Description

This function applies community detection to a graph and plots the result.

Usage

```
Community.Detection.Plot(network)
```

Arguments

network	A graph object.

Value

A plot visualizing the graph with nodes colored according to their community membership. The plot also displays the modularity score as a sub-title, indicating the strength of the community structure.

Examples

```
## Not run:
  # Assuming 'network' is a predefined graph object
  # network <- make_graph("Zachary")
  Community.Detection.Plot(network)

## End(Not run)
```

f3 *Persian Text Normalization and Stemming*

Description

This function normalizes Persian text by replacing specific characters and applies stemming.

Usage

```
f3(x)
```

Arguments

x	A character vector of Persian text.

Value

Returns a character vector where each element is the normalized and stemmed version of the corresponding element in the input vector. Specifically, it performs character replacement and stemming on each element of the input, thereby returning a vector of the same length but with processed text. If an element cannot be processed, it will be returned as NA in the output vector.

Examples

```
## Not run:
  text <- c("Persian text here")
  normalized_text <- f3(text)

## End(Not run)
```

f5 *Filter Data Frame by Document ID*

Description

This function filters a data frame by the specified document ID. If the ID is 0, the entire data frame is returned.

Usage

```
f5(UPIP, I)
```

Arguments

UPIP	A data frame with a column named 'doc_id'.
I	An integer representing the document ID.

Value

Returns a subset of the input data frame (‘UPIP‘) containing only the rows where the ’doc_id’ column matches the specified document ID ‘I‘. If ‘I‘ is 0, the function returns the entire data frame unmodified. The output is a data frame with the same structure as the input but potentially fewer rows, depending on the presence and frequency of the specified ID.

Examples

```
data <- data.frame(doc_id = 1:5, text = letters[1:5])
filtered_data <- f5(data, 2)
```

---

f6 *Extract Token Information from Data Frame*

Description

This function extracts token, lemma, and part-of-speech (POS) tag information from a given data frame and compiles them into a new data frame.

Usage

```
f6(UPIP)
```

Arguments

UPIP	A data frame containing columns ’token’, ’lemma’, and ’upos’ for tokens, their lemmatized forms, and POS tags respectively.

Value

Returns a new data frame with three columns: ’TOKEN’, ’LEMMA’, and ’TYPE’. ’TOKEN’ contains the original tokens from the ’token’ column of the input data frame. ’LEMMA’ contains the lemmatized forms of these tokens, as provided in the ’lemma’ column. ’TYPE’ contains POS tags corresponding to each token, as provided in the ’upos’ column. The returned data frame has the same number of rows as the input data frame, with each row representing the token, its lemma, and its POS tag from the corresponding row of the input.

Examples

```
data <- data.frame(token = c("running", "jumps"),
                   lemma = c("run", "jump"),
                   upos = c("VERB", "VERB"))
token_info <- f6(data)
```

f7 *Extract and Count Specific Parts of Speech*

Description

This function extracts tokens of a specified part of speech (POS) from the given data frame and counts their frequency.

Usage

```
f7(UPIP, type)
```

Arguments

UPIP	A data frame with columns 'upos' (POS tags) and 'lemma' (lemmatized tokens).
type	A string representing the POS to filter (e.g., 'NOUN', 'VERB').

Value

Returns a data frame where each row corresponds to a unique lemma of the specified POS type. The data frame has two columns: 'key', which contains the lemma, and 'freq', which contains the frequency count of that lemma in the data. The rows are ordered in decreasing frequency of occurrence. This format is useful for quickly identifying the most common terms of a particular POS in the data.

Examples

```
data <- data.frame(upos = c('NOUN', 'VERB'), lemma = c('house', 'run'))
noun_freq <- f7(data, 'NOUN')
```

fun.all.sums *Apply Suffix Modifications to Persian Words*

Description

This function iteratively applies a series of suffix modifications to a vector of Persian words.

Usage

```
fun.all.sums(v, TYPE = TYPE.org)
```

Arguments

v	A character vector of Persian words.
TYPE	A vector of suffix types for modification.

Value

Returns a character vector where each element corresponds to a word from the input vector ‘v‘ with all specified suffix modifications applied. This results in a transformed vector where each word has been modified according to the series of suffix types provided in ‘TYPE‘. The length of the returned vector matches the length of the input vector.

Examples

```
## Not run:
  words <- c("Persian text here")
  modified_words <- fun.all.sums(words, TYPE)

## End(Not run)
```

`fun.one.sums` *General Persian Suffix Modification*

Description

This function modifies Persian words based on a specified suffix type.

Usage

```
fun.one.sums(v, type)
```

Arguments

`v`	A character vector of Persian words.
`type`	A character string representing the suffix type.

Value

Returns a character vector where each element corresponds to a word from the input vector ‘v‘ with the specified suffix type modified. This results in a transformed vector where each word has been modified to remove or alter the specified suffix. The length of the returned vector matches the length of the input vector, and each word is modified independently based on the specified suffix type.

Examples

```
## Not run:
  words <- c("Persian text here")
  modified_words <- fun.one.sums(words, "Persian text here")

## End(Not run)
```

FUNbigrams *Extract Bigram Information and Count Frequency*

Description

This function processes a data frame containing bigrams and their frequency, and creates a new data frame with separated words and their frequencies.

Usage

```
FUNbigrams(tf.bigrams)
```

Arguments

tf.bigrams	A data frame with bigram terms and their frequency.

Value

A tibble data frame where each row represents a unique bigram from the input data. The data frame contains three columns: 'word1' and 'word2' representing the individual words in the bigram, and 'weight' representing the frequency of the bigram in the corpus. This structure facilitates further analysis of the bigram relationships and their occurrences.

Examples

```
tf_bigrams <- data.frame(term = c("hello_world", "shiny_app"),
                         term_freq = c(3, 2))
bigram_info <- FUNbigrams(tf_bigrams)
```

fungan *Persian Suffix Modification for 'Persian text here' Suffix*

Description

This function modifies Persian words ending with 'Persian text here' suffix.

Usage

```
fungan(v)
```

Arguments

v	A character vector of Persian words.

Value

Returns a character vector where each element corresponds to a word from the input vector ‘v‘ with the ’Persian text here’ suffix modified. This results in a transformed vector where each word ending with the specified suffix is altered. The length of the returned vector matches the length of the input vector, and each word is modified independently based on the presence of the specified suffix.

Examples

```
## Not run:
  words <- c("Persian text here")
  modified_words <- fungan(words)

## End(Not run)
```

`fungi` *Persian Suffix Modification*

Description

This function modifies Persian words ending with ’Persian text here’ suffix.

Usage

```
fungi(v)
```

Arguments

`v`	A character vector of Persian words.

Value

Returns a character vector where each element corresponds to a word from the input vector ‘v‘ with the specified suffix modified. This results in a transformed vector where each word ending with the specified suffix is altered. The length of the returned vector matches the length of the input vector, and each word is modified independently based on the presence of the specified suffix.

Examples

```
## Not run:
  words <- c("Persian text here")
  modified_words <- fungi(words)

## End(Not run)
```

funmi *Modify Persian Words Starting with 'Persian text here'*

Description

This function modifies Persian words starting with the prefix 'Persian text here'.

Usage

```
funmi(v)
```

Arguments

v	A character vector of Persian words.

Value

Returns a character vector where each element corresponds to a word from the input vector 'v' with the specified suffix modified. This results in a transformed vector where each word ending with the specified suffix is altered. The length of the returned vector matches the length of the input vector, and each word is modified independently based on the presence of the specified suffix.

Examples

```
 ## Not run:
  words <- c("Persian text here")
  modified_words <- funmi(words)

## End(Not run)
```

LEMMA *Persian Lemmatization*

Description

This function performs lemmatization on a vector of Persian words.

Usage

```
LEMMA(Y, TYPE = TYPE.org)
```

Arguments

Y	A character vector of Persian words.
TYPE	A vector of suffix types for modification.

Value

Returns a character vector where each element is the lemmatized form of the corresponding element in the input vector ‘Y‘. Lemmatization involves removing inflectional endings and returning the word to its base or dictionary form. The length of the returned vector matches the length of the input vector, and each word is lemmatized independently based on the specified suffix types in ‘TYPE‘.

Examples

```
## Not run:
  words <- c("Persian text here")
  lemmatized_words <- LEMMA(words, TYPE)

## End(Not run)
```

`network.cor` *Create and Plot a Correlation Network*

Description

This function creates a correlation network based on specified terms and a threshold, and optionally plots it.

Usage

```
network.cor(dt, Terms, threshold = 0.4, pl = TRUE)
```

Arguments

`dt`	A document-term matrix.
`Terms`	A vector of terms to check for correlation.
`threshold`	A numeric threshold for correlation.
`pl`	A logical value to plot the network or not.

Value

If 'pl' is TRUE, a plot of the correlation network is displayed, highlighting the strength of associations between terms. If 'pl' is FALSE, a data frame with correlation pairs and their corresponding weights is returned.

PMI *Calculate Pointwise Mutual Information (PMI)*

Description

This function calculates the PMI for collocations in a given text data.

Usage

```
PMI(x)
```

Arguments

x	A data frame with columns 'token' and 'doc_id'.

Value

Returns a data frame where each row represents a unique keyword (collocation) in the input data. The data frame contains columns such as 'keyword', representing the keyword, and 'pmi', representing the PMI score of that keyword. Higher PMI scores indicate a stronger association between the components of the collocation within the corpus.

Examples

```
data <- data.frame(token = c("word1", "word2"), doc_id = c(1, 1))
pmi_scores <- PMI(data)
```

ScaleWeight *Scale a Numeric Vector*

Description

This function scales a numeric vector by a specified lambda value.

Usage

```
ScaleWeight(x, lambda)
```

Arguments

x	A numeric vector.
lambda	A numeric scaling factor.

Value

A numeric vector where each element of the input vector 'x' is divided by the scaling factor 'lambda'. This results in a scaled version of the input vector.

Examples

```
scaled_vector <- ScaleWeight(1:10, 2)
```

`server` *Server Logic for MadanText Shiny Application*

Description

This function contains the server-side logic for the MadanText application. It handles user inputs, processes data, and creates outputs to be displayed in the UI.

Usage

```
server(input, output)
```

Arguments

`input`	List of Shiny inputs.
`output`	List of Shiny outputs.

Value

This function sets up the reactive environment and output elements in the Shiny application. It does not return any value but modifies the Shiny app's UI based on user inputs and reactive expressions. It returns a Shiny Server object.

`set.graph` *Set Graph Attributes*

Description

This function sets various attributes for a given graph object, including vertex degree and edge width.

Usage

```
set.graph(network)
```

Arguments

`network`	A graph object.

Value

The input graph object with added attributes: 'degree' for each vertex and 'width' for each edge. These attributes enhance the graph's visual representation and analytical capabilities.

ui *User Interface for MadanText*

Description

This function creates a user interface for the MadanText Shiny application. It includes various input and output widgets for file uploads, text input, and visualization.

Usage

```
ui
```

Format

An object of class `shiny.tag.list` (inherits from `list`) of length 4.

Value

A Shiny UI object.

Index